COMPUTER AIDED NESTING OF IRREGULAR SHAPES

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

JAGDISH MAHESHWARI

to the

DEPARTMENT OF MECHANICAL ENGINEERING
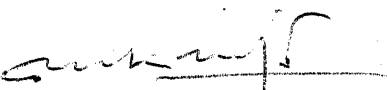INDIAN INSTITUTE OF TECHNOLOGY KANPUR
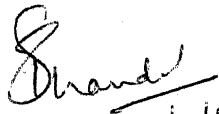MARCH, 1987

ME-1987-M-MAH-COM

CERTIFICATE

This is to certify that the work entitled "Computer Aided Nesting of Irregular Shapes" by Jagdish Maheshwari has been carried out under our supervision and has not been submitted elsewhere for the award of a degree.

Dr. M.K. Muju
Professor
Dept. of Mech. Engg.
I.I.T. Kanpur

Dr. S.G. Dhande
Professor
Dept. of Mech. Engg. and
Comp. Sc. and Engg.
I.I.T. Kanpur

ACKNOWLEDGEMENTS

I am deeply indebted to Prof. S.G. Dhande and
Prof. M.K. Muju under whose supervision and guidance I could
complete this thesis work successfully. There were moments
of anxiety but their encouragement and appreciation carried
me through this task.

I am grateful to the CAD Project staff for rendering
me every possible help in this work.

I am thankful to Mr. R. Shukla for nicely taking
photographs of the displays of the results.

In the end, I wish to express my thanks to Mr. R.N.
Srivastava for his excellent typing of the manuscript.

JAGDISH MAHESHWARI

CONTENTS

LIST OF FIGURES

NOMENCLATURE

$a[i]$ = Area of shape i

f_i = Objective function for separating tiny shapes

$\bar{f}$ = Mean value of f_i

j_i = Any piece

$l[i]$ = Length of shape i

$l_c[i]$ = Length of 'candidate shape' i

l_s = Total length of allocation

$l*$ = Length parameter of the current sub-problem to be solved

$n[i]$ = Number of unallocated pieces of shape i

$n_c[i]$ = Number of unallocated pieces of 'candidate shape' i

$n_s[i]$ = Number of pieces of 'selected shape' i

r = Total number of shapes in 'candidate shapes set'

t = Total number of irregular shapes for automatic nesting

$w[i]$ = Width of shape i

$w_c[i]$ = Width of 'candidate shape' i

$w*$ = Width parameter of the current sub-problem to be solved

σ = Standard deviation

A_i = An instantaneous allocation of a piece

CS = Candidate shapes set

DS = Set of shapes to be allocated automatically on the stock

K = Weightage variable

LS = Length of stock

P = Solution set for 'selected shapes'

SS   =   Selected shapes set

U   =   Union operation

WS   =   Width of stock

ABSTRACT

The problem of allocating a specified number of two-dimensional regular or irregular different pieces on a stock of rectangular shape having finite dimensions is of relevant interest to some industrial processes. The problem finds applications in ship-building, garment, leather industries, sheet metalcutttthggeetc.

An approach is given in the present work for optimal layout of two-dimensional irregular shapes using a heuristic method. The algorithm produces an approximate solution which proves to be of good quality and efficient in terms of computer time. It works by hierarchically decomposing original problem into many sub-problems such that solution of each will imply a good solution of the original problem. The proposed methodology has been implemented in the form of a menu-driven programmenon OMEGA 58000 system. Several examples were tested using the software developed.

CHAPTER I

INTRODUCTION

1.1 Introduction to Nesting Problems

The problems of allocation of regular and irregular shapes arise frequently in applications where it has to be determined as to how a set of two dimensional pieces will fit into (or be cut from) a large stock sheet of finite dimensions, the object being to maximise the value of pieces fitted (or cut) or alternatively minimise the waste. This optimisation problem has been solved for a long time manually which needed all the more time if the number of shapes were large. Such problems appear, for example, in blanking parts from strips, cutting a leather piece for making handbags, handgloves, jacket, shoes etc., cutting a cloth piece for garmetas, cutting of metal sheets for sheet-metal applications (contour cutting of sheets). Other applications include cutting of rectangular sheets of steel, wood, glass etc., cutting of paper board for production of boxes. Ship building and railway coach making also involve cutting of irregular shapes from large sheets of metal.

The method of nesting shapes on a stock must be connected to the technological cutting ways which follow it. Indeed, these methods induce some constraints on the relative position of the shapes on the stock. On the other hand, according to the application, the material can show some anisotropic

properties. In some applications, the large stock to be used may contain some defective areas and it is required that allocated pieces must not overlap these defects. Depending upon the application other constraints like spatial, position, orientation and adjacency may also be specified along with the main problem.

A broad spectrum of the problem is shown in Figure 1. Here a distinction has been made between problems of strip layout for blanking applications and stock layout for leather, cloth, sheet etc. cutting primarily on the basis of their different problem domains and different approaches for their solutions.

Most of the strip layout problems involve single component whose orientation on the strip is more important than its allocation position. On the other hand stock layout problems may involve more than one component. Solution approach requires that position as well as orientation of the component on the stock be determined for which wastage is minimum.

Stock layout problems may be further sub-divided into trim-loss and knapsack problems (Hinxman [1]). In a knapsack problem each of the ordered pieces is given a value and the *objective* ~~object~~ is to maximize the total value of the pieces to be cut from a stock item. In the trim-loss problem, on the other hand, there is a set of integers $\{d_1, d_2, \ldots, d_n\}$, called the order list, which describes the number of pieces of each *shape* ~~size~~. The *objective* ~~object~~ is to minimise the total cost of the stock items consumed in satisfying the orders.

Nesting Problems

```
Nesting Problems
        |
  ┌─────┴─────────────────────────┐
  |                                |
Strip Layout                  Stock Layout
(blanking                     (leather, cloth, sheet-metal
applications)                 cutting applications)
                                   |
                          ┌────────┴────────┐
                          |                 |
                      Trim-Loss         Knapsack
                          |
                 ┌────────┴────────────────┐
                 |                         |
           Rectangular               Irregular
           Shapes                    Shapes
                 |                    (all shapes
                 |                    other than
                 |                    rectangular)
        ┌────────┴────────┐
        |                 |
  Non-guillotine      Guillotine
```
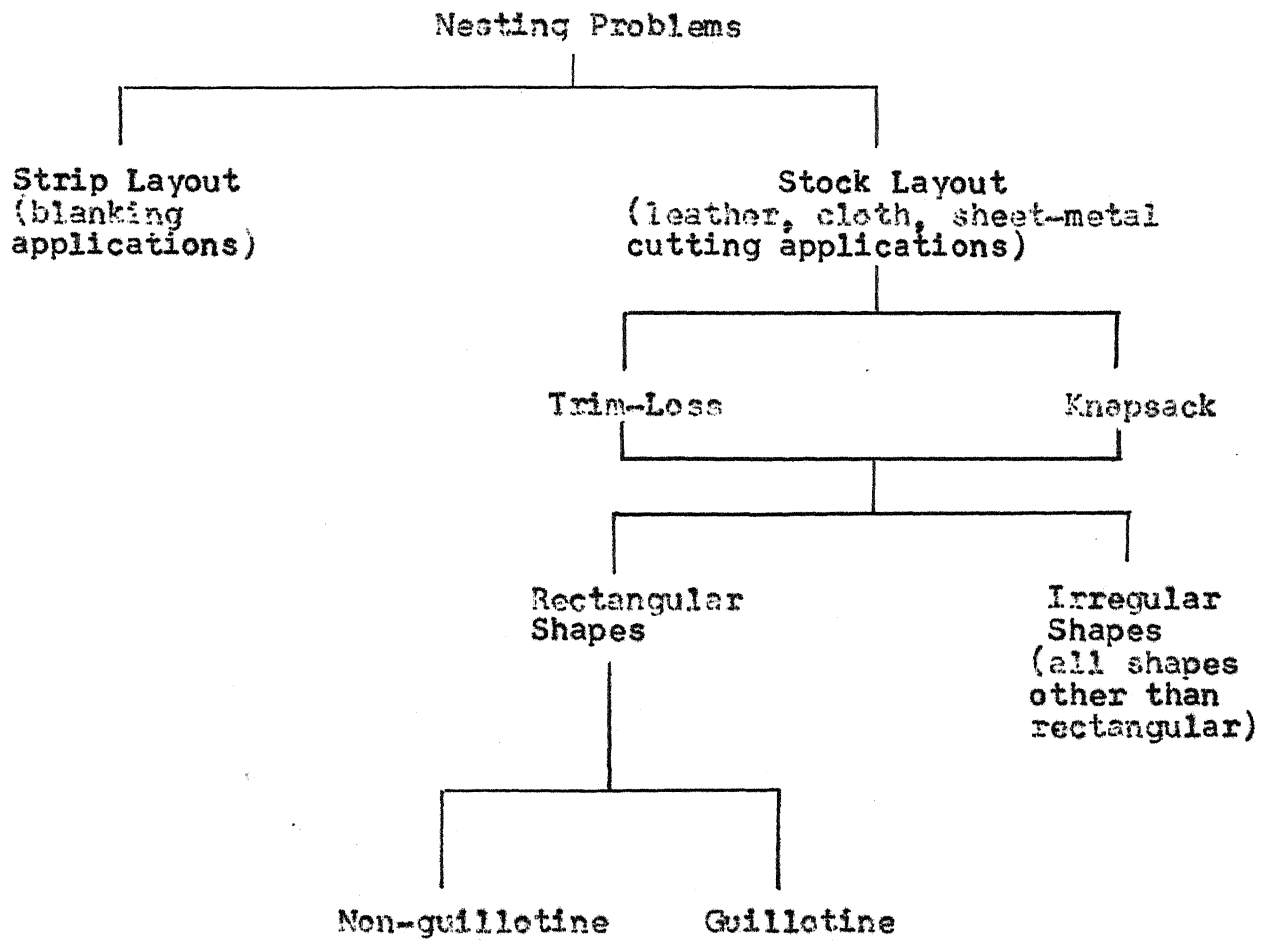
Fig. 1.   Spectrum of Nesting Problems.

Further, both trim-loss and knapsack problems can be defined for rectangular as well as irregular shapes and solution approaches in both the cases would be different.

Depending upon the application, rectangular shape cutting may be of guillotine or non-guillotine type. A guillotine cutting requires that a cut is made from one edge to the opposite edge of the stock along a straight line. In non-guillotine cutting the cut does not extend right through.

1.2 Review of the Literature

Many researchers have contributed to the solution of nesting problems in different ways and in various domains.

Chow [2] has found an approach for nesting shapes in a single or double row on a flat strip. He has presented three different approaches. AYC Nee [3] has also reported a simple microcomputer based strip layout solution for single row layout and pairwisely clustered layout.

Rectangular-shape cutting stock problem has been studied by many researchers. Gilmore and Gomory [4, 5] have resolved the rectangular cutting stock problem by the way of linear programming using the algorithm of knapsack problem which applies to a guillotine cutting of the material in order to give rectangular shapes. Christofides and Whitlock [6] presented a tree-search algorithm for the rectangular cutting stock problem in which there exist constraints on the maximum number of each type of piece and all cuts are of guillotine type. Their algorithm, however, is inefficient in terms of computer time when

solving problems of large size as are often met in practical applications. Adamowicz and Albano [7] found an approach for guillotine cutting using dynamic programming. Albano and Orsini [8] improved their algorithm and presented their own proposal of a heuristic tree-search approach to the knapsack problem for guillotine cutting. The algorithm is efficient in terms of computer time for computing an approximate solution close to the optimal. Beasley [9] has presented both heuristic and exact algorithms based on dynamic programming for staged as well as non-staged version of the problem. This algorithms are capable of dealing with large problems.

Admowicz and Albano [10] have studied the nesting of irregular geometrical shapes on a rectangular stock. The method of solution proposed consists of enclosing irregular shapes, singly or in combination in minimum area enclosing rectangles called 'modules'. The modules are then packed onto the given rectangular stock using a dynamic programming algorithm. The effectiveness of the algorithm was shown by comparing the results with those obtained by using manual methods. However, there are other applications in which the kind of solution produced is not satisfactory because it is not possible to include in the algorithm a number of constraints specific to the problem. Albano [11] therefore later proposed a system where a tentative solution is generated and then interactive improvements are allowed. Although these investigations have been successful in handling specific subsets of the general optimal two-dimensional allocation problem, their approach,

however, has resulted in procedures that are marginally useful for applications with a small number of pieces where an exact ordering of the pieces is required.

Albano and Sapuppo [12] have used a heuristic search methods for two-dimensional layout of irregular shapes on rectangular stock sheet. The problem is reduced to the search of an optimal path in a graph and, using a heuristic search method, an approximate solution which proves to be of good quality and efficient in terms of computer time can be obtained.

In their approach a proper choice of cost estimate function $\hat{h}(n)$ in evaluation function $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$ is very important as it has direct bearing on the amount of time it takes for the solution. Here $\hat{h}(n)$ is an estimate of expected waste and it should always be lower than the waste that would result in the optimal solution if the piece in question were to be included in the solution. If $\hat{h}(n)$ is a good approximation to the expected waste then the algorithm will execute fairly efficiently most of the time otherwise more search is required. And if $\hat{h}(n)$ is an overestimate then it may lose the optimal path. A proper choice of $\hat{h}(n)$ is however rather difficult to decide and therefore need for other method arises.

1.3 Objective and Scope of the Present Work

The objective of the present work is to design and develop an interactive, adaptive method for optimal arrangement of irregular shapes using heuristic techniques. Number of pieces of each shape to be cut are also specified. The

solution of the problem should minimise wastage or, in other words, maximise density of packing.

The problem has been studied in 'the form which was considered by Albano and Sapuppo [12] with some significant diversions in solution approach. Limitations of the cost estimate function $\hat{h}(n)$ which is used in their work has been discussed in Section 1.2. In the present work this limitation is overcome by adopting a different approach for the solution.

The computer program developed is interactive and user-friendly. Procedures for automatic as well as interactive nesting have been implemented whereby a user can arrange shapes on stock interactively like a draftsman does.

Present work falls in the category of trim-loss problems having applicability for both irregular as well as rectangular shapes. It has limited scope for strip layout problems. Only spatial constraints have been considered in the present work.

1.4 Organisation of the Work

The present work has been organised in the following manner.

Chapter II is an introduction to heuristic techniques, their scope and limitations. It describes two heuristic techniques which are related with the present problem.

Chapter III describes the problem in detail and explains the complete methodology of the solution.

Chapter IV discusses the software details of the implementation and explains how to use the program.

Results are discussed in Chapter V. Some limitations of the present work are also included in this chapter.

Chapter VI concludes the work by presenting a summary of it and throws light upon the various modifications and expansions of present approach which would make it more useful from practical point of view.

CHAPTER II

HEURISTIC TECHNIQUES

2.1 Introduction to Heuristic Techniques

Heuristics means a method which on the basis of experience or judgement seems likely to yield a good solution to a problem but which cannot be guaranteed to produce the most optimum one.

In order to solve many hard problems efficiently it is often necessary to compromise the requirements of mobility and systematicity and to construct a strategy that is no longer guaranteed to find the best answer but that will almost always find a very good answer. This is the idea of the heuristic. Heuristic techniques improve the efficiency of a search process possibly by sacrificing claims of completeness. Heuristics are like tour guides. They are good to the extent that they point in interesting directions; they are bad to the extent that they can lead into dead ends. Using good heuristics one can hope to get good solutions (even if non-optimal) to hard problems in reasonably good time.

2.2 Scope and Limitations of Heuristic Techniques

In general a heuristic technique will be highly domain dependent, that is, it will use information about the particular problem for which it is developed in order to find solutions. So it may have little or no applicability beyond that particular

problem. Even apparantly similar problems may require radically different heuristic technique. Nevertheless, there are some good general purpose heuristics that are useful in a wide variety of problem domains.

In the following ~~two~~ section, two important heuristic approaches are discussed which are used for Artificial Intelligence problem solving. The first approach has been adopted for the present work, while Albano and Sapuppo [12] have used the second approach in their work.

2.3 Heuristic Techniques Used in Artificial Intelligence

2.3.1 The Nearest Neighbour Approach

Consider the following problem.

The Travelling Salesman Problem

Statement: A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow so that he travels the shortest possible distance on a round trip, starting at any one of the cities and then returning to the same.

The 'nearest neighbour method' works by selecting the locally superior alternative at each step. Applying it to the travelling salesman problem produces the following procedure:

1. Arbitrarily select a starting city.

2. To select the next city, look at all cities not yet visited. Select the one closest to the current city. Go to it next.

3. Repeat step 2 until all cities have been visited.

If there are N number of cities then the nearest neighbour method takes time proportional to N squared while a tree-search method would take time proportional to N! and a branch-and-bound would take an average time proportional to e^N.

It has been shown that it is possible to prove an upper bound on the error the nearest neighbour approach incurs. Rich [15] has shown that in the worst case when the choices are pessimally ordered, from worst to best, the ratio between the answer found using the nearest neighbour method and the optimal answer is less than or equal to (lg N + 1)/2.

In the average case when the cities are distributed at random, Rich [15] has shown that this approach finds tours that are about 20% worse than optimal.

Details of application of this approach to the nesting problem will be discussed in Section 3.5.6.2.

2.3.2 State Space Search Approach

Many problems in Artificial Intelligence and operations research are solved by a general problem solving technique based on search through a space of candidate solutions (Nillson [18]).

The first step consists in associating to the problem a set of states and operators transforming one state into another. The set of states reachable from the initial state can be seen as a direct graph containing nodes corresponding to the states and areas corresponding to the operators. With this representation

in mind, the solution is seen as a search process for finding a path from the initial to any member of a set of nodes called the goal or final nodes.

The algorithm followed for this search is given below.

1) Put the start node on a list called GENERATED.

2) If GENERATED is empty, exit with failure; otherwise continue.

3) Select a node from GENERATED according to a rule R and put it on a list called EXPANDED. Call it n.

4) If n is a goal node, exit with the solution path; otherwise continue.

5) Expand n, that means generate all its successors. If there are no successors, go to step 2; otherwise put them on GENERATED, and go to step 2.

CHAPTER III

DEVELOPMENT OF INTERACTIVE PROGRAM

3.1 Problem Description and Solution Approach

In Chapter I the broad spectrum of the nesting problems, objective and scope of the present work have been discussed. This section describes the problem in detail and enunciates the solution approach. The solution approach makes the following assumptions:

1. The shapes to be arranged are simply connected polygons without holes.

2. The stock is rectangular and large enough to contain all the demanded pieces of all shapes.

3. The shapes can be rotated and allocated anywhere on the stock.

4. The preferred orientation for any shape is one in which the longest edge of the minimum area enclosing rectangle for the shape is along the length direction of the stock.

5. Shapes having same areas but different lengths and widths are treated as different.

The constraints which have been considered are:

1. Any allocation of a shape should not exceed beyond given length and width of the stock or a sub-part of it.

2. Shapes should not overlap one another.

An interactive computer program has been developed which is semi-automatic in nature. It interacts with user to accept description of shapes and stock; finds out tiny shapes

from this set of user defined shapes; suggests a layout automatically as close to the optimal as possible for all big shapes and then allows user to interactively arrange all remaining tiny shapes inside waste pockets or in the unallocated space on the stock. Shapes are classified as big and tiny using a statistical analysis which is similar to the ABC classification of items in inventory management.

The automatic solution approach for big shapes is based on certain heuristic rules and method. It resolves the original problem of allocating shapes on a stock having length and width as LS and WS respectively into a number of sub-problems having different lengths and widths such that solution of each sub-problem will produce an optimal solution.

The software consists of four modules for handling functions of geometric representation and orientation, determination of tiny shapes, automatic nesting of all big shapes and interactive arrangement of tiny shapes. All the modules are user-friendly and menu driven.

The main algorithm for the solution is given below.

Main Algorithm

Step 1:   Accept geometric description of shapes and their
         ordered pieces;

Step 2:   Orient shapes;

Step 3:   if user wants to separate tiny shapes then

             - interact with user to find tiny shapes and store
               them in list TINY;

- store remaining big shapes in list BIG;

and if

else

- store all shapes in list BIG;

Step 4:   Automatic nesting of all shapes in list BIG;

Step 5:   if list TINY is not empty then do

- interactive placement of shapes in list TINY;

Iff the following sections all the steps are described
separately in detail. The organisation is as follows:

Section 3.2: Geometric Representation

Section 3.3: Orientation of Shapes

Section 3.4: Determination of Tiny Shapes

Section 3.5: Automatic Nesting

Section 3.6: Interactive Placement.

3.2 Geometric Representation

Geometry of all the irregular shapes to be allocated
and of the resource is approximately represented by polygons
whose vertices are stored in terms of (x, y) coordinates. The
polygonal representation has its strength in solving problems
involving tracing of line or polygon boundaries, computation of
area, perimeter etc. Moreover graphics output devices can be
efficiently used by inputting coordinates of vertices of the
polygon. Such a representation also demands that all curves and
arcs be approximately represented as polygons.

The two-dimensional geometry of shapes and resource is
described by straight line segments passing between vertices
(x_i, y_i), $i = 1, 2 \ldots N$ and $(x_N, y_N) = (x_1, y_1)$.

A software package '2D-Modeller' has been implemented
for interactively creating any two-dimensional irregular shape.
Its details are given in Chapter IV.

3.3 Orientation of Shapes

All the shapes are oriented in such a way that the
length of the minimum area enclosing rectangle for each shape
is preferably along the length direction of the stock. A minimum
area enclosing rectangle is determined for each shape (Dhande
and Rumulu [14]) and then this rectangle is rotated in the manner
as described above. Finally this angle of rotation is used for
orienting the shape. Otherwise, user may also specify an angle
of rotation for each shape. The program interacts with user to
seek his/her acceptance for the preferred orientation, otherwise
prompts him/her to input the angle of rotation of his/her choice.

3.4 Determination of Tiny Shapes

While arranging irregular shapes on a given stock,
wastage is unavoidable. This wastage can be utilized by inter-
actively arranging tiny shapes inside them. An approach similar
to the ABC classification of items in inventory management has
been adopted here to determine tiny shapes.

Finding out of tiny shapes from a set of shapes which
are to be allocated on a given stock requires some formal mathe-
matical basis. One criteria for this may be to compare areas of
all the shapes and the shapes which are below some lower bound
area are classified as tiny. Since the approach treats two

shapes having same areas but different lengths and widths as distinct shapes, another criteria may be length too.

For finding out tiny shapes an objective function f_i is defined as

$$f_i(\ ) = K * l[i] + (1 - K) * a[i] \quad \text{for all } i \leqslant t;$$

where

t = total number of shapes to be allocated,

and $\quad 0 \leqslant K \leqslant 1.0;$

Here $l[i]$ and $a[i]$ are length and area of shape i respectively.

The variable K is called the 'weightage variable'. Value of K $(0 \leqslant K \leqslant 1.0)$ is specified by user when the program is executed.

For each shape, value of objective function f_i is evaluated and a mean value $\overline{f}$ is computed as follows:

$$\overline{f} = \frac{\sum\limits_{i=1}^{t} f_i * n[i]}{\sum\limits_{i=1}^{t} n[i]};$$

where $n[i]$ = number of pieces of shape i.

The standard deviation σ is given by:

$$\sigma = \sqrt{\frac{\sum\limits_{i=1}^{t} n[i] * (f_i - \overline{f})^2}{\sum\limits_{i=1}^{t} n[i]}};$$

This value of the standard deviation becomes the basis for separating tiny shapes from big shapes. Those shapes whose

f_i is less than $(\bar{f} - 6)$ are classified as tiny and they are not considered for automatic nesting procedure. This implies that in the present work about 16% of the shapes will be classified as tiny shapes. This is based on the assumption that f_i is normally distributed.

3.5 Automatic Solution Approach

3.5.1 Terminology

In this section some terms are defined in connection with the present approach for the solution of the problem.

1) Length of an Irregular Shape: It is the maximum distance between two extreme vertices of the polygonal approximation of the shape along the length direction of the stock.

2) Width of an Irregular Shape: It is the maximum distance between two vertices of the polygonal approximation of the shape along the width direction of the stock.

3) Sub-Problem: At any stage of the solution an allocation of any shape has to meet the constraints on length and width of the stock or a sub-part of it. A sub-problem represents a stock or a sub-part of it and specifies new length and width within which shapes are to be allocated. In the beginning these length and width parameters of the sub-problem are initialised as length and width of the stock.

4) Candidate Shapes Set [CS]: It is the set of all those unallocated shapes which satisfy the length and width constraints as prescribed by a sub-problem under examination and can possibly be allocated on the stock.

$$CS = \left\{ (l_c[i], \, w_c[i], \, n_c[i]), \, (l_c[i+1], \, w_c[i+1], \, n_c[i+1]), \right.$$
$$\left. \cdots (l_c[r], \, w_c[r], \, n_c[r]) \right\},$$

where with $l_c[r] \geqslant L_m * l_c[i].$

$l_c[i]$, $w_c[i]$ and $n_c[i]$ are respectively length, width and number of unallocated pieces of candidate shape i,

and

r = number of 'candidate shapes'.

The shapes contained in the set CS are called 'candidate shapes'.

5) <u>Selected Shapes Set [SS]</u>: It is the set of the number of pieces ($n_s[i]$) of a 'candidate shape' i in the set CS which can feasibly be allocated in the given width of the stock or a sub-part of it, as prescribed by the current sub-problem being solved.

$$SS = \left\{ n_s[i], \, n_s[i+1], \, \ldots, \, n_s[r] \right\},$$

where

$$0 \leqslant n_s[i] \leqslant n_c[i] \quad \text{for all } i \leqslant r;$$

Here each shape i in the set CS whose $n_s[i] > 0$ is called a 'selected shape'.

3.5.2 Description of the Procedure

~~The approach for the solution is a combination of heuristic and exact techniques.~~ The solution is found by a sequential decision making process based on certain heuristics that first finds out all 'candidate shapes' and then selects shapes from this set of 'candidate shapes' which can be feasibly

allocated on the stock satisfying all constraints on length and width. Then applying a placement policy, the most optimum allocations for these 'selected shapes' are found using the 'nearest neighbour approach'. After all 'selected shapes' have been allocated, the procedure replaces the original problem by two new sub-problems and decides whether they can be feasibly filled by remaining unallocated shapes. During the process on the current sheet at most three parts can be distinguished, the part that is already filled, the part currently under examination and the part still available for allocation. A new sub-division will not affect the first part but it reduces the size of the second. After a sub-problem is solved the next sub-problem is invoked. This process will continue as long as there are pieces to be allocated.

The main algorithm developed for the automatic solution is given below.

The algorithm is based on the following main heuristics:

1. Once an acceptable layout of the shapes has been found for stock or a sub-part of the stock, this will not be reconsidered, in other words, the remaining decisions do not influence a previously solved sub-problem or previously allocated shapes.

2. The stock or a sub-part of the stock is always filled along its width first. This heuristic is very important in reducing the dimensions of the problem. In fact as it will be seen in subsequent sections, at each stage the two-dimensional allocation problem is reduced to one-dimensional

whereby shapes are allocated in the width direction of the stock or a sub-part of it.

3. For a stock or a sub-part of it, always the longest pieces are chosen first for finding their possible allocations.

Use of last two heuristics is supported by the fact that it is adopted by an experienced man to solve this kind of problem by hand.

Automatic Nesting Algorithm

1. Input stock parameters and initialisation;

2. Sort all shapes in list BIG in order of decreasing length;

3. While BIG is not empty do
    - generate sub-problems and push them in stock GENERATED;
    - pop next sub-problem to be solved from GENERATED;
    - find 'candidate shapes set' for the current sub-problem;
    - find 'selected shapes set' for the current sub-problem and store them in list SELECTED;
    - find optimum allocations for all the 'selected shapes' in list SELECTED;
    - modify list BIG;
    end while.

In the following sections each step of the automatic nesting procedure is explained in detail.

3.5.3 Generation of Sub-Problems

The definition of the sub-problem was already presented in Section 3.5.1. The following paragraphs now explain its generation.

Whenever allocation of all the shapes of a 'selected shapes set' is complete, two new sub-problems are generated. This procedure is explained with the help of Figure 2.. The figure shows a stock and some 'selected shapes' allocated on it.

In the beginning when 'no shape has been allocated, the length and width parameters of the current sub-problem to be solved are length (LS) and width (WS) of the stock respectively and the space available for arrangement of irregular shapes is a rectangle. After arranging all the 'selected shapes', the space available for further allocation is reduced to that shown by hatched area.

This available space is divided into two parts, 1 and 2, as shown in the figure and each part is associated with a new sub-problem referred to as sub-problem no. 1 and sub-problem no. 2 respectively.

If $w*$ and $l*$ are length and width respectively of the current sub-problem being solved, then length and width parameters of sub-problem no. 1 will be l_s and $w* - w_s$. Whereas length and width parameters of sub-problem no. 2 will be $l* - l_s$ and $w*$ respectively (Figure 2). Here l_s is the length of the currently placed selected shape having the maximum extension to right and w_s is the width of the stock consumed for this allocation.

Parameters of these sub-problems are stored in a stack structure. Therefore the next sub-problem to be solved is invoked on the basis of last-come-first-served. The sub-problem no. 1 always has precedence over no. 2.
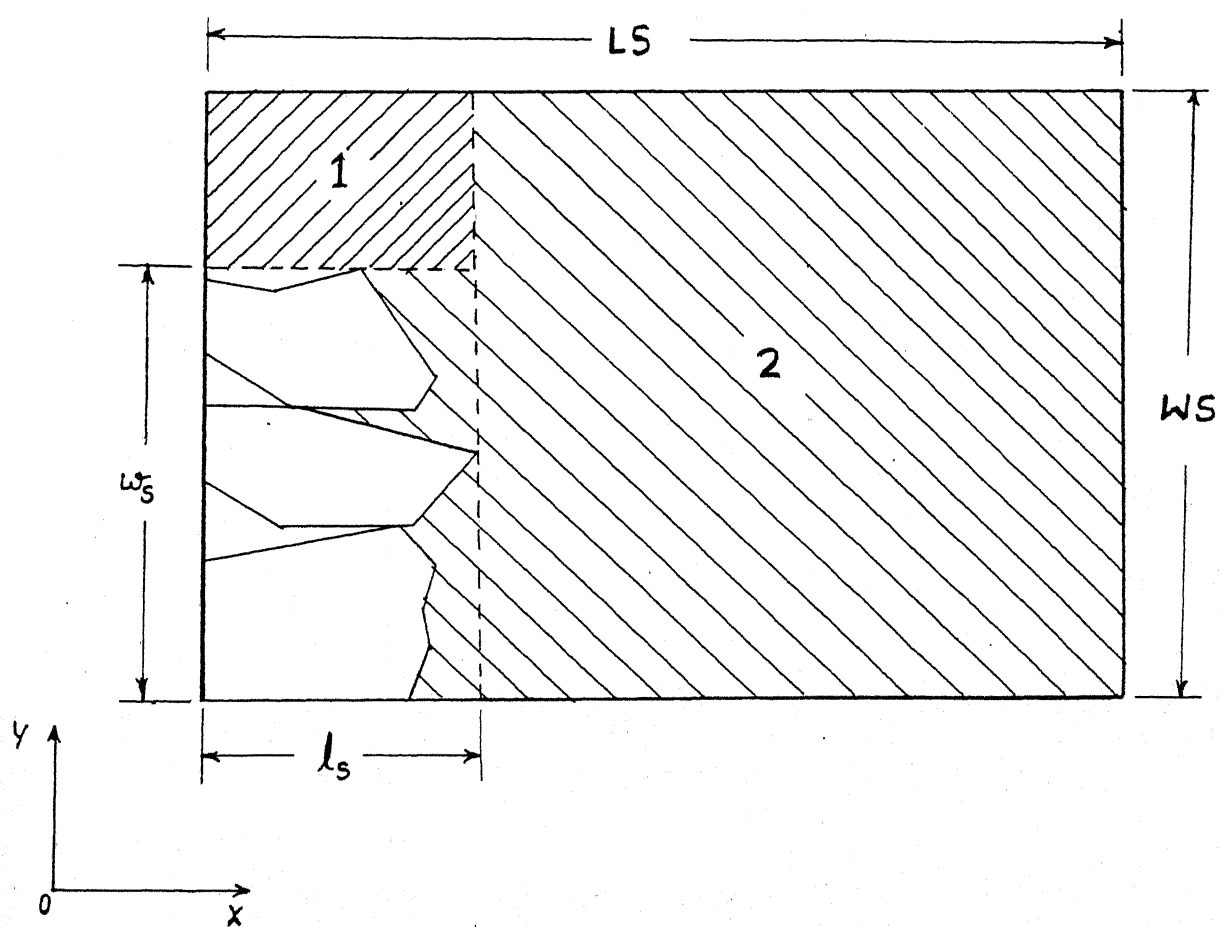
Figure 2.   Generation of Sub-Problems

If it is not possible to find even a single piece which satisfies the required length and width constraints of an invoked sub-problem then that sub-problem is omitted and the next one is invoked; otherwise 'candidate shapes set' and 'selected shapes set' are determined for that sub-problem. The method and criteria for these are explained in the next sections.

3.5.4 Determination of 'Candidate Shapes Set'

All irregular shapes in 'candidate shapes set' have to satisfy constraint on length l* as well as width w* as prescribed by the sub-problem being solved. Let the demanded set of unallocated shapes for automatic nesting be:

$$DS = \left\{ (l[1], w[1], n[1]), (l[2], w[2], n[2]), \ldots \right.$$
$$\left. \ldots (l[t], w[t], n[t]) \right\},$$

where

$$l[i] \geq l[i + 1] \text{ for all } i \leq t;$$

Here

l[i], w[i] and n[i] are respectively length, width and number of unallocated pieces of shape i,

and

t = total number of shapes in the set DS.

A set of 'candidate shapes' [CS] is obtained from the set DS by the following steps.

Step 1: Select the shape in the set DS with $n[i] > 0$ which satisfy the following conditions

$$l[i] \leq l* \text{ and } w[i] \leq w*;$$

Here $l[i]$ is called 'reference-length' and shape i is called 'first-candidate'.

A lower bound is defined as L_m * reference-length. It was decided to assign a value of 0.9 to L_m after investigating its influence on the layout produced.

Step 2: Select all the shapes in the set DS after the 'first-candidate' with $n[i] > 0$ which satisfy the following condition

$l[i] \geqslant L_m$ * reference-length;

If $l[i] \geqslant (L_m$ * reference-length) and if $l[i + 1] < (L_m$ * reference-length) then shape i is called 'last-candidate'.

Step 3: Include all the shapes between and including the 'first-candidate' and the 'last-candidate' in the 'candidate shapes set'.

fc = first-candidate,

lc = last-candidate.

$$CS = \left\{ (l_c[fc], w_c[fc], n_c[fc]), (l_c[fc + 1], w_c[fc + 1], n_c[fc + 1]), \ldots (l_c[lc], w_c[lc], n_c[lc]) \right\}.$$

The cardinality of the set CS denoted by $|CS|$ is defined as the number of elements in CS. Let

$$|CS| = r.$$

3.5.5 Determination of 'Selected Shapes Set'

In Section 3.5.1 the 'selectedsshapes set' was defined as

$$SS = \left\{ n_s[i], \; n_s[i + 1], \; \ldots \; n_s[r] \right\},$$

where

$$0 \leqslant n_s[i] \leqslant n_c[i] \quad \text{for all } i \leqslant r.$$

If w* is the width parameter of the current sub-problem then all the 'selected shapes' in SS have to satisfy the following inequality:

$$\text{maximise}(\sum_{i=1}^{r} w_c[i] * n_s[i]) \leqslant w^* \qquad \ldots \; (1)$$

The problem of finding a possible combination of the 'candidate shapes' in the set CS (having width as $w_c[i]$) and their individual number of instances ($n_s[i]$) which satisfy (1) is solved by employing a tree-search procedure. Albano and Orsini [13] have reported the solution of a similar problem known as M-partitions problem using a tree-search method. Their algorithm has been used here after making some modifications relevant to the present work in it.

A multiset $S_r = \left\{ s[i] \right\}$ is defined whose elements are again a set as shown below:

$$s[i] = \left\{ w_c[i], \; n_c[i] \right\} \quad \text{for all } 0 < i \leqslant r.$$

Here $w_c[i]$ and $n_c[i]$ are width and number of pieces of a 'candidate shape' i and r is the cardinality of the set S i.e. $|S| = r$.

Let B be a binary tree (Figure 3) with $g = |S| + 1$ prime-levels. Each prime-level 'g' in turn will have sub-levels 'h' where $h \leqslant n_c[i]$. Each node of the tree at any level except

the leaves have two sons corresponding to the inclusion or exclusion of $w_c[i]$ of the ith shape of the set S_r into the solution. Each level of the tree is denoted by a number 'g.h' as shown in the figure.

With any node of the tree is associated:

1) the path from the root up to it represented by a solution set

$$P = \left\{ p[i] \right\},$$

where

$$p[i] = x[i] * n_s[i],$$
$$x[i] \in \left\{ 0, 1 \right\},$$

and

$$0 \leqslant n_s[i] \leqslant n_c[i] \quad \text{for all} \quad 0 < i \leqslant r;$$

2) a partial sum defined by

$$\sum_{k=1}^{i} p[k] * w_c[k].$$

Every node of the tree at any level stores the partial sum as defined above up to that level. The search is stopped at any level of the tree as soon as this partial sum becomes equal to w* and the procedure returns the status of the solution set P at that level. If it is not possible to find a suitable combination of the 'candidate shapes' and their individual instances whose summation of widths (inequality (1)) is exactly equal to w* then the procedure always returns a solution which is nearest to the desired solution.

In Figure 3 an example of the tree-search procedure as described above is illustrated for $S_2 = \{(6, 3), (3, 2)\}$. In the present example there are three prime-levels, $g = |S| + 1 = 3$. The prime-level no. 1 in turn will have three sub-levels while no. 2 will have only two sub-levels. Inclusion or exclusion of $w_c[i]$ of the ith shape of the set S_2 in the solution is shown by associating with arcs, 1 and 0 respectively.

The solution set $P = \{p[1], p[2]\}$ is initialised to null in the beginning. With every instance of inclusion of $w_c[i]$ of the ith element of S_2 in the solution, the corresponding $p[i]$ in P is incremented by one and the corresponding $n_c[i]$ in S_2 is decremented by one. With this representation in mind the solution sets for different values of w* are given below:

1. $w* = 15$; $P = \{2, 1\}$; Figure 4.1.

2. $w* = 22$; $P = \{3, 1\}$; Figure 4.2.

In the last example, the attempt is to find a solution which is nearest to the desired solution.

A recursive algorithm for the solution can be derived as follows:

PROP(i, m, P, T~~PROP~~,

where

    i = index of next $s[i]$ to be examined;

    m = the remaining sum: w* - (sum of all the $w_c[i]$ used
       upto this point);

    $P = \{j; s[j]$ is included in the path from the root to
       the current node$\}_c$

Figure 3 Tree-Search Procedure

1·1

6 (0)

1
1·2

6 (6)

1
1·3

6 (12)

0
2·1

3 (12)

1
2·2

(15)

0
3·1

(15)

Figure 4·1

1·1

6 (0)

1
1·2

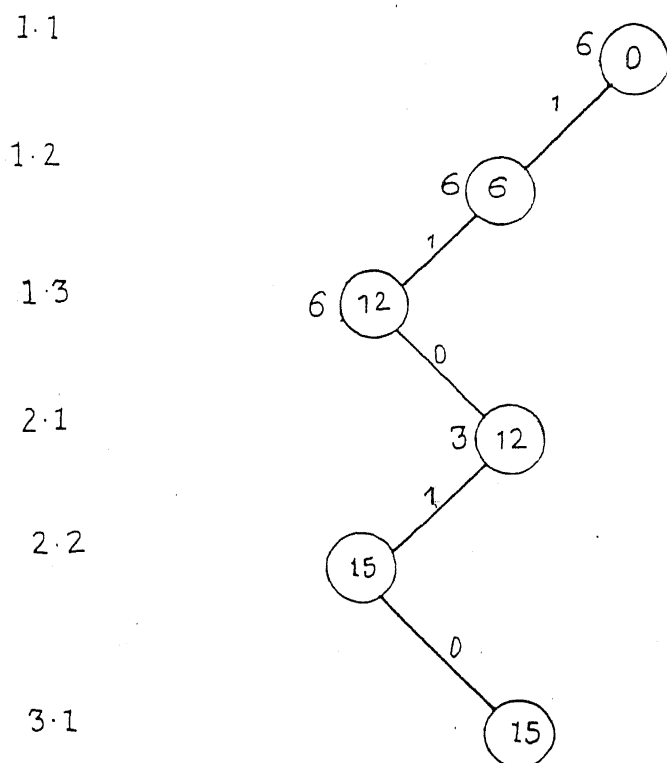6 (6)

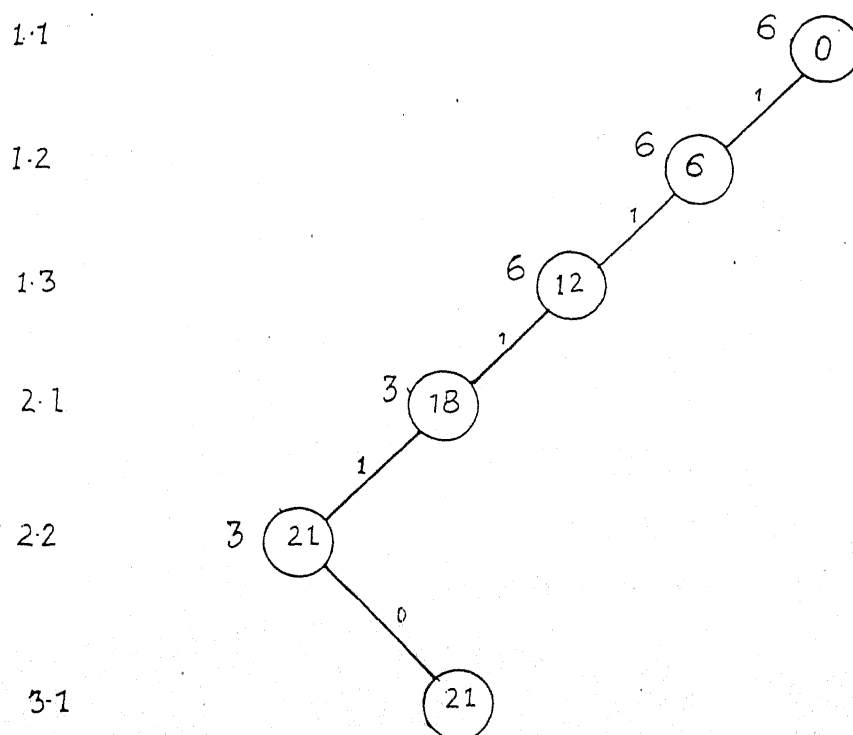1
1·3

6 (12)

1
2·1

3 (18)

1
2·2

3 (21)

0
3·1

(21)

Figure 4.2 Search Paths

c = number of instances of $s[j]$ included in the path from the root to the current node;

T = remaining $n_c[i]$ in S_r;

$$SUM(i) = \sum_{j=i}^{r} w_c[j] \text{ in } S_r;$$

It is initially invoked as PROP(1, w*, NULL, $n_c[1]$).

Algorithm

1. [All done] if $i > r$ then output P;

2. [Test heuristics] if $SUM(i) \leq m$ then output

   $$P \cup \{i, i + 1, \ldots, r\}_{n_c[i]};$$

   return;

3. [Try to include $s[i]$]

   if $w_c[i] < m$ then

       if $T > 1$ then

           $T = T - 1$;

           $P = P \cup \{i\}_1$;

           PROP(i, $m - w_c[i]$, P, T)

       end if

     else

       $P = P \cup \{i\}_1$;

       $T = n_c[i + 1]$;

       PROP(i + 1, $m - w_c[i]$, P, T)

     end else

   end if

   else

       if $w_c[i] = m$ then

$$\text{output } P \cup \{i\}_1;$$

$$\text{return}$$

$$\text{end if}$$

$$\text{else}$$

$$\text{if } \sum_1^{n_c[i]} w_c[i] = m \text{ then}$$

$$\text{output } P \cup \{i\}_{n_c[i]};$$

$$\text{return}$$

$$\text{end else}$$

4. $\left[\text{Exclude } s[i]\right]$ if $w_c[i] > m$ then

$$T = n_c[i + 1];$$

$$\text{PROP}(i + 1, m, P, T);$$

5. $\left[\text{Return}\right]$ return.

3.5.6 Allocation of Selected Shapes

Allocation of all the 'selected shapes' of a set always proceeds along the width direction of the resource i.e. stock, in such a manner that for each allocation the wastage is minimum. This requires that each piece be placed in close contact with the profile of the resource. In the following sections the placement policy and the allocation procedure are discussed in this respect.

3.5.6.1 Placement Policy: The 'allocation profile' of the resource is the 'front' produced by the rightmost pieces which are exposed to the part of the resource not yet used. Figure 5 provides examples of profiles indicated by hatched lines.
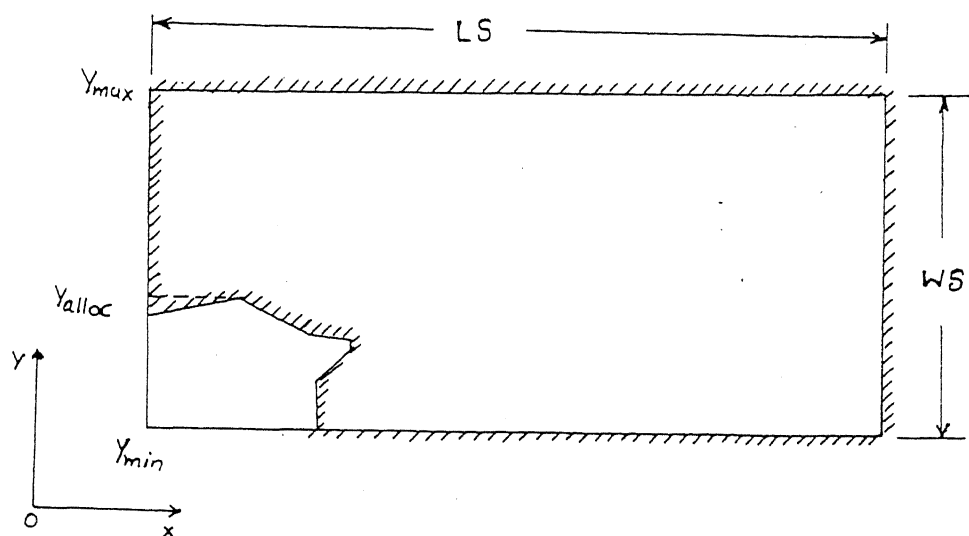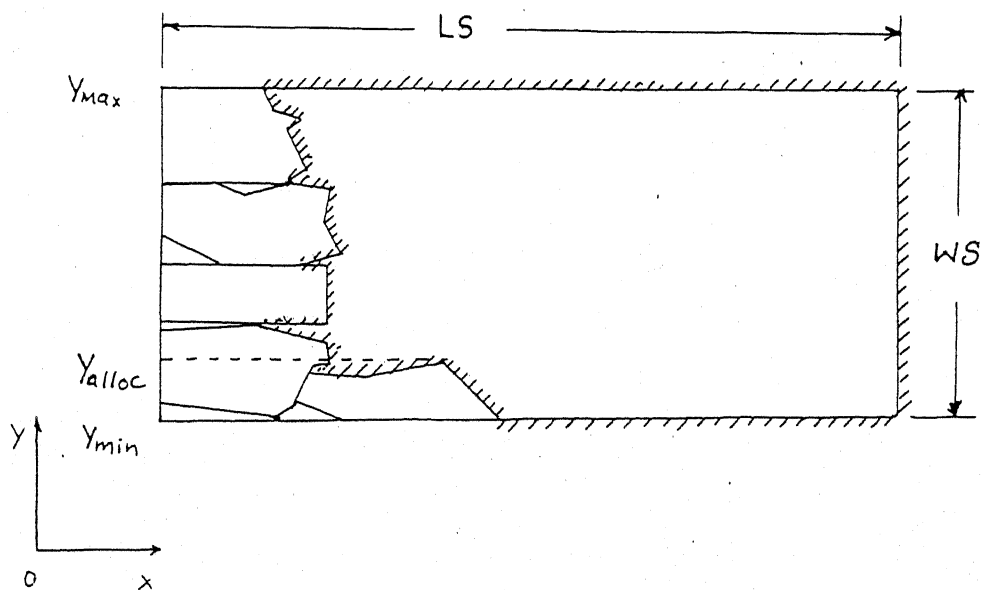
Figure 5.1.

Figure 5.2 Allocation Profiles

All the selected shapes are allocated in accordance with a prefixed placement policy. The placement policy has to satisfy all the required constraints on the layout.

The policy states that the allocation of pieces always starts from the lower most edge (y_{min}) of the stock and proceeds in upward direction till the upper most edge (y_{max}) is reached (Figure 5(i)). A piece is always allocated adjacent to the last allocated piece at a position which produces the left most allocation (Figure 5(ii)). After every allocation, the 'allocation profile' of the resource is modified to include the right most profile of the last allocated piece. The policy ensures that all allocated pieces are clubbed together with minimum waste space between them.

Computational Efficiency

In order to improve efficiency of computation without losing optimality of the solution very much, it is proposed that next allocation always occurs at the maximum y coordinate of the last allocated shape as indicated by y_{alloc} in Figure 5. This allocation position along the y axis is specified for each subproblem too when they are generated and whenever a new subproblem is invoked, the next allocation always begins from that allocation position.

3.5.6.2 **Allocation Procedure:** Let A_i denotes an instantaneous allocation of a piece j_i on the resource. Then an allocation A_{i+1} is a successor of A_i if it is obtained by arranging

one of the unallocated pieces in A_i according to the placement policy.

Given an instantaneous allocation A_i, let the space available, space (A_i), be the area on the right side of the profile i.e. the area which may be used to allocate a piece j_i. As an example, in Figure 6 the area a2 will not be considered usable any more. Then the 'added waste' in A_i as a consequence of the allocation of j_i is defined as

$$\text{added waste } (A_i) = \text{area of space } (A_{i-1})$$
$$- \left\{ \text{area of space } (A_i) + \text{area of } j_i \right\};$$

After all the shapes have been allocated, the wastage produced is computed by two methods.

1. Relative Waste: It is given by

$$\text{Relative waste} = (WS*l_s - \text{area of all the allocated pieces})/WS*l_s;$$

WS and l_s are respectively the width of the resource and the length of the layout, that is, in the final allocation the whole area between the pieces and the rectangle with sides equal to the given width of the stock and a length sufficient to wholly include all the pieces, will be considered as waste. Albano and Sapuppo [12] have proposed this method of finding the waste.

2. Absolute Waste: It is computed as

$$\text{Absolute waste} = \text{waste } (A_i)/WS*LS,$$

where

$$\text{Waste } (A_0) = 0,$$

and

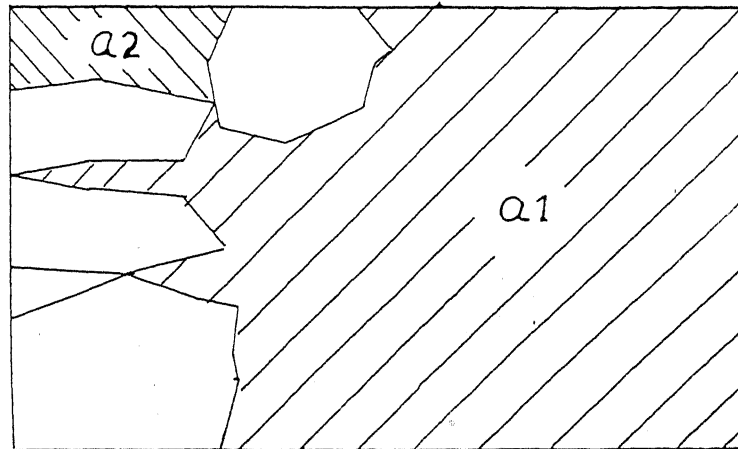Figure 6 Wastage Example

Waste (A_i) = Waste (A_{i-1}) + Added waste (A_i).

The area of the irregular polygonal shapes and the resource is computed as follows (Wilson and Farror [16]).

The geometry of all the shapes and the resource is represented by line segments passing between vertices (x_i, y_i), $i = 1, 2, \ldots, N$ and $(x_N, y_N) = (x_1, y_1)$. Then their areas are given by

$$\text{Area} = -\sum_{i=1}^{N} F_i(y_i + 0.5 * G_i),$$

where

$$F_i = X_{i+1} - X_i,$$

and $G_i = Y_{i+1} - Y_i.$

The optimal allocation problem is transformed into the search of a 'selected shape'; given 'allocation profile' of the resource and position along the y axis at which next allocation will occur for which the 'added waste' is minimum. This optimality test is carried out for two orientations 0° and 180° of each 'selected shape'.

Nearest neighbour approach has been adopted for finding allocations for each selected shape. The algorithm to implement it is given below.

Algorithm for Allocation of Selected Shapes

While list SELECTED is not empty do
- for each shape ~~piece~~ in list SELECTED not yet placed
  - for each orientation of 0° and 180°
    - allocate piece tentatively;

```
            - find 'added waste';
        end for
    end for
        - select shape for which 'added waste' is minimum;
        - allocate the piece finally;
        - modify the allocation profile of the resource;
        - store new ordinate position for next allocation;
        - modify list SELECTED;
    end while.
```

3.6 Interactive Placement of Tiny Shapes

Whenever an irregular shape is allocated on a stock, some part of the stock goes waste. The profile of this waste part of the stock is stored after every allocation. Here a simple heuristic is applied which decides whether to store this profile or not. The area of the waste part is compared with the minimum area of all the tiny shapes and if it is found greater tthan the profile is stored. Figure 7 shows three waste parts 1, 2, 3 and their profiles in hatched lines.

Interactive features of this program allow user to locate any waste part on the screen with the help of a locating device, then pick up any tiny shape and moved it on the screen with the help of the locating device. The user can also rotate the shape by any angle. Here interactive part is like a tool which assists user in rotating and moving any tiny shape on the screen. It does not help the user in deciding which shape to pick, which can be feasibly allocated inside a waste pocket. It indicates to the user whenever any shape overlaps other shapes.
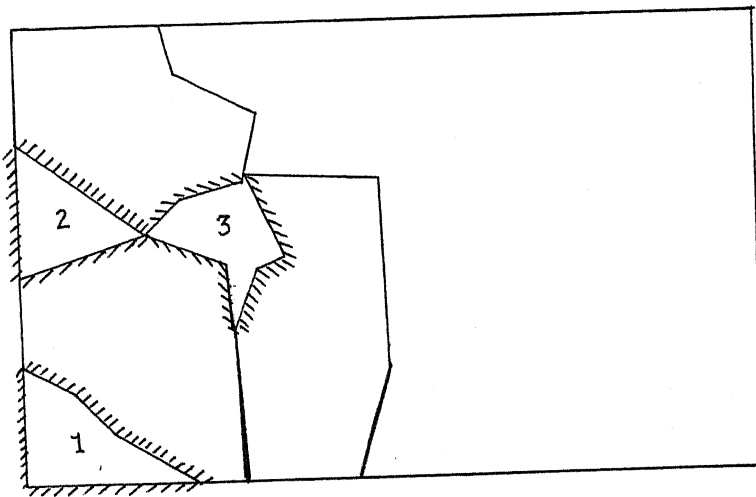
Figure 7. Waste Part Profiles

CHAPTER IV

IMPLEMENTATION DETAILS

4.1 Software Details

The program for the present work has been written in 'C' language on 'Omega 58000' system. Advantages of using 'C' language are that the program execution is faster because it is a relatively 'low level' language and it is very well supported by Unix operating system. Moreover it is independent of any particular machine architecture and therefore portable programs can be written easily.

For graphics input and output system routines 'wgread', 'wline' and 'wdbyte' have been used. Graphics output is displayed in windows which are created and run under the window management system. For menu creation, display and command picking 'pickmenu' system routine has been used. The interactive placement of tiny shapes has been implemented using raster operations for which 'wrasop' system call is invoked.

The program is user-friendly and includes suitable prompts and error messages to guide user in running it. On line help has been provided with each menu display. The program includes a '2D-Modeller' for creating any two-dimensional polygonal geometrical shape interactively on the screen. The geometry may consist of straight lines, horizontal lines, vertical lines and arcs of a circle. Details of it and how to use the main program are given in the next two sections.

4.2 2D-Modeller

This program helps user in defining a two-dimensional
shape which can be created interactively on the graphics screen.
Procedures to draw straight lines, horizontal lines, vertical
lines, arc of a circle given its centre and subtended angle and
arc through three points have been implemented. User can
select any of these procedures by selecting the corresponding
command from a menu. Arcs are approximated by straight line
segments. An error message is displayed in window 'Error
Window' whenever user commits any mistake.

All the shapes should always be defined in anticlockwise
direction. Arc of a circle given its center and angle is also
drawn anti-clockwise. Maximum number of sides in a shape should
not exceed 30. Details of how to use this package are given in
the next section.

4.3 How to Use the Program

The program should be run under the Window Management
System. The program can be invoked by typing 'a.out'. A
prompt message is displayed which prompts user to press button
A on the puck for displaying the top level menu. This menu
has following commands available.

1. Profile Generation and Processing: Using this command,
two-dimensional geometrical shapes can be defined and data
stored in a file or if the data is already available in a file
then they can be read from that file. Each created shape is
called a segment. Maximum number of shapes which can be created

is 20. Three windows i.e. 'Display Window', 'Prompt Window' and 'Error Window' are created on the screen. When this command is selected another menu is displayed in which following commands are available.

(A) Create Segment: It is for defining two-dimensional polygons ~~gons~~ interactively on the screen using a locating device. The polygons can have upto 30 sides only. User can select any of the following commands from the menu which is displayed when the command is selected.

(i) Move: This is for selecting the starting point. User can locate and then select a point on the screen by pressing button B. No point is selected if any other button is pressed. Coordinate position of the locating device is also indicated on the screen. It is to be noted that first command should always be 'Move' command for defining any new shape.

(ii) Line: A line is drawn from the last point to the currently selected point whenever button A is pressed. More than one point can be selected but at the last point, button B should be pressed to quit the command.

(iii) Arc: It is for creating an arc of a circle. Input parameters are center of the arc and its angle. A point for the center can be selected by pressing button B. A message will prompt user to input angle of the arc which should be in degrees. The arc is converted into straight line segments.

(iv) Arc-3Pt: It is for creating an arc through three points. Only two points need to be defined. Button A should be pressed for the first point and B for the second one. Again the arc is

converted into straight line segments. Arc may be in any direction depending upon positions of points.

(v) <u>H-line</u>:   It is for drawing a horizontal line through the last point. Button B should be pressed.

(vi) <u>V-line</u>:   It is for drawing a vertical line through the last point. Again button B should be pressed.

(vii) <u>Polygon</u>:   A side of a polygon is drawn whenever button A is pressed. When button B is pressed the polygon is closed automatically and the command is quit.

(viii) <u>Delete</u>:   The last line drawn can be deleted by selecting this command.

(ix) <u>ClLine</u>:   It is for closing a polygonal shape by a straight line.

(x) <u>Quit</u>:   It is for quitting the 'Create Segment' command.

In case, if a wrong command is selected then that command can be quit prematurely by locating the cursor inside a small window in the left top corner of the 'Display Window' followed by pressing of button A.

(B) <u>Store Segment</u>:   Every time a new polygonal shape is created, it should be stored by selecting this command. Here user will be prompted to give his/her choice of any orientation of the created shape other than the preferred orientation. User should respond with 'y' if he/she wants otherwise 'n'. The default response is 'n'.

(C) <u>Clear</u>:   It clears the 'Display Window'.

(D) <u>Write</u>:   When this command is selected it creates a default

file 'Data' if it does not exist or deletes its contents when
it already exists. All the relevant data pertaining to descrip-
tion of a shape like its length, width, area, number of pieces,
x and y coordinates of the vertices, indexes of maximum and
minimum x, y coordinates are stored in the file 'Data' in that
order.

(E) Read: It is opposite to 'Write' command described above.
The default file 'Data' containing description of shapes can
be directly read using this command and then 'Nesting Automatic'
command, described below, can be selected.

(F) Delete Segment: It deletes a segment from the list of
stored segments when its segment number is input.

(G) Delete All: It deletes all the stored segments.

2. Nesting Automatic: This command is for automatic nesting
of shapes. User is prompted to type 'y' if he/she wants to
separate tiny shapes from the automatic nesting procedure other-
wise 'n'. Here default is 'y'. If user's response is 'y' then
a prompt message will ask him/her to type value of the weightage
variable K which should not be more than 1.0 (see Section 3.4).
After doing the statistical analysis, if the procedure finds
tiny shapes, they will be separated and all big shapes will be
displayed in 'Nesting Window'. User should press return to
continue.

3. Nesting Interactive: When this command is selected all
tiny shapes with their number of unallocated pieces and areas
are displayed on the screen. User can pick any tiny shape for

arranging it interactively by typing its number which is same
as number of the window whiwhich that tiny shape is displayed.
A menu is displayed when button A is pressed. During the inter-
activepplacement of a shape if that shape overlaps any other
allocated shape then their points of intersections will become
invisible on the screen giving an indication to user about the
overlap.

(A) Fill Waste: Any valid waste pocket can be filled using
this command. The waste pocket can be selected on the screen
itself by positioning the cursor inside it followed by pressing
of button A. If it is a valid waste pocket then its area will
be displayed. A menu is displayed when button A is pressed.
Using 'Pick' command of this menu, any tiny shape can be picked
and then moved anywhere on the screen by moving the puck while
keeping button A on it pressed. Button C should be pressed in
order to quit the command 'Pick'. If placement is not satisfac-
tory then that tiny shape can be erased from its position by
selecting 'Erase' command or conversely if it is satisfactory
then 'Store' command should be selected. The latter command
decreases number of unallocated pieces of that tiny shape by
one and displays it on the screen in its window.

(B) Interactive Placement: After filling all waste pockets,
user can selectthhis command if any tiny shape is still unallo-
cated. Using this command he/she can arrange a shape in the
unallocated space of the stock. There is no great distinction
between this command and the previous command except that in the
previous case user will have an option to either store or erase

image of an interactively arranged tiny shape. In this command
an allocation of any shape will always be stored when this
command is quit. Using both the commands a shape can be
arranged anywhere inside the stock irrespective of the waste
pocket which was selected for filling.

CHAPTER V

RESULTS AND DISCUSSION

5.1 Experimental Results

The program was run to find optimal pattern layouts for rectangular as well as irregular shapes. Their results are discussed below. Photographs of the layouts are ~~appended~~ affixed for each example. Here the numbers shown in the photographs correspond to the hierarchical number of the successive sub-problems which were generated in each case. The pieces which are not numbered are those which were declared tiny by the program and arranged interactively. Results of all the layouts are given in Table 1.

Figure 8 shows the layout obtained for rectangular shapes. Figures 9 and 10 are the layouts for shirt and trousers patterns respectively. The descriptions of these patterns were obtained from the Ordnance Parachute Factory, Kanpur. Figure 11 shows a marked improvement in the layout of trousers patterns when the interactive facility of arranging tiny shapes is utilized.

Figures 12 and 15 provides examples of layouts for two different cases, generated using the pieces considered by Albano and Sapuppo [12] (Figures 14 and 17) for their results. A comparison between their results and results obtained by adopting the present approach is given in Table 1. Exact figures cannot be given because the piece descriptions were obtained from Albano and Sapuppo's layouts.
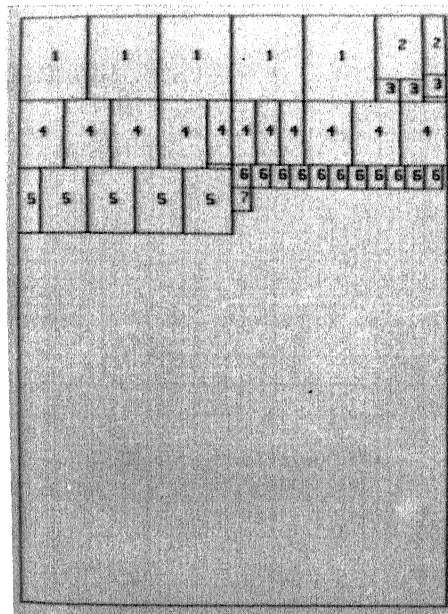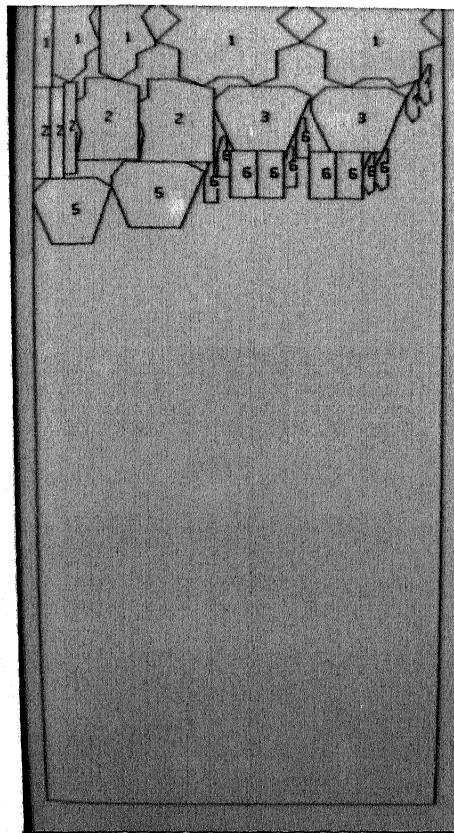
Figure 8   Nesting of Rectangular Shapes
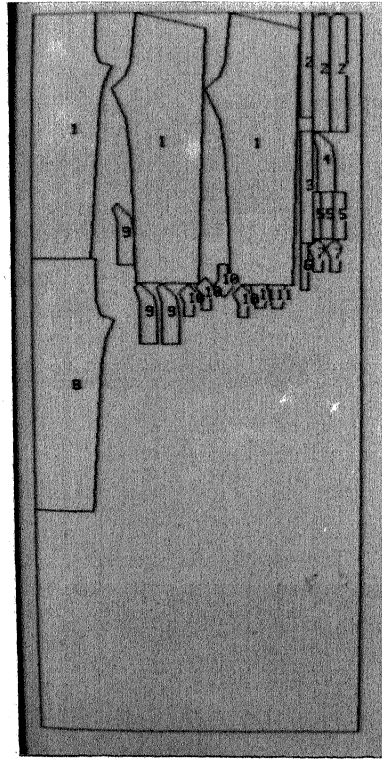
Figure 9. Nesting of Shirt Patterns
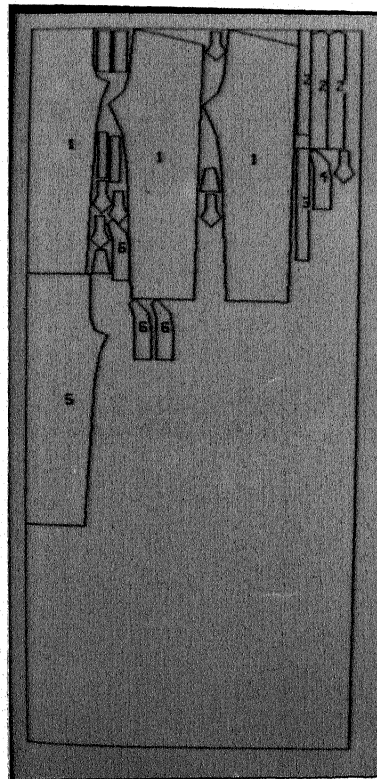
Figure 10 Nesting of Trousers Patterns (automatic)

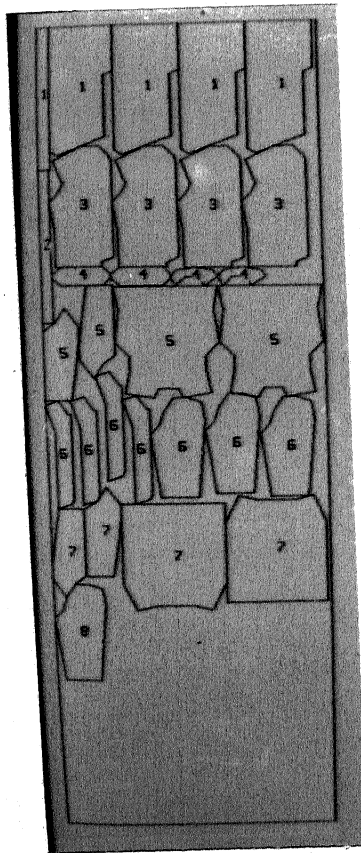Figure 11 Nesting of Trousers Patterns (semi - automatic)

Figure 12 Albano and Sapuppo's Case No. 1 Using the present Approach (automatic)
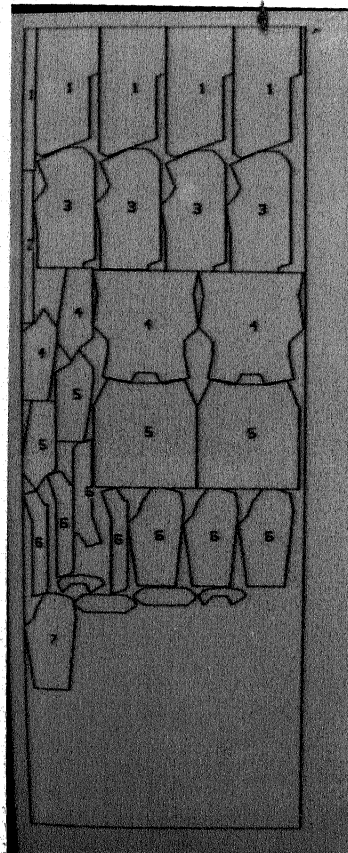

Figure 13 Albano and Sapuppo's Case No. 1 Using the present Approach (semi-automatic)
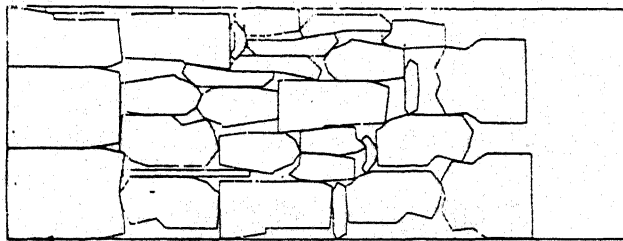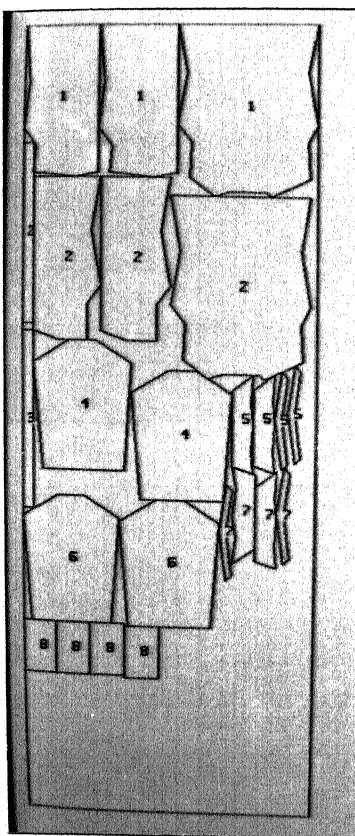
Figure 14 Albano and Sapuppo's Case No. 1

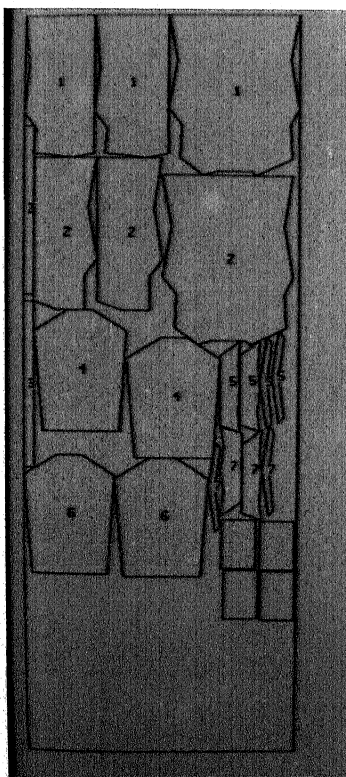Figure 15 Albano and Sapuppo's Case No 2 Using the Present Approach (automatic)

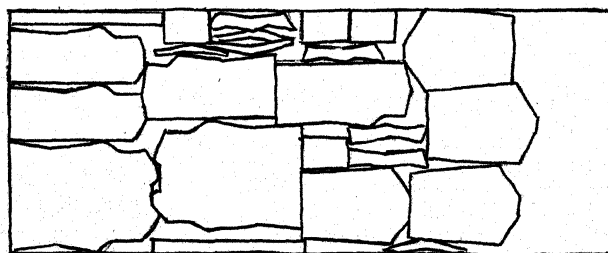Figure 16 Albano and Sapuppo's Case No 2 Using the Present Approach (semi-automatic)

Figure 17 Albano and Sapuppo's Case No. 2

TABLE 1

Results of Layouts

S. No.	Figure number	Types of shapes	TA	Solution using the present approach					Albano & Sapuppo's solution	
				Were tiny shapes separated?	Value of K	AW %	RW %	CT	RW %	CT
1	2	3	4	5	6	7	8	9	10	11
1	8	Rectangular	38	No	—	0.0	6.0	316	NA	NA
2	9	Shirt patterns	26	No	—	15.0	28.0	162	NA	NA
3.1	10	Trousers patterns	25	No	—	15.0	43.0	80	NA	NA
3.2	11	Trousers patterns	25	Yes	1.0	7.0	44.0	30	NA	NA
4.1	12	Albano & Sapuppo's case no. 1 (Figure 14)	30	No	—	11.0	22.0	140	19.4	1206
4.2	13	Albano & Sapuppo's case no. 1 (Figure 14)	30	Yes	0.95	7.0	23.0	126		
5.1	15	Albano & Sapuppo's case no. 2 (Figure 17)	24	No	—	9.0	18.0	82	17.4	426
5.2	16	Albano & Sapuppo's case no. 2 (Figure 17)	24	Yes	1.0	5.0	18.0	66		

TA = Total number of pieces to be allocated
AW = Absolute waste
RW = Relative waste
NA = Not available

CT = Computation time is proportional to
K = The weightage variable to separate tiny shapes

It can be seen by referring to the Table 1 that figures of the 'Relative Waste' in both the cases using the present approach are quite near to those of Albano and Sapuppo. On the other hand present approach is more efficient than their approach. The computation time it takes is proportional to the summation of the square of the number of pieces at the same hierarchical level which are automatically allocated by the computer. For instance there are 5 pieces at hierarchical level 1 while at level 2 there is only one piece (Figure 12). On the other hand the computation time in Albano and Sapupo's approach is proportional to the number of generated nodes. These figures are also given in the Table 1 in the columns 9 and 11.

5.2 Limitations of the Methodology

The methodology adopted in the present work is based on certain heuristic rules which suggest good solution in most of the cases in reasonably good computational time. Nevertheless it has some limitations which are always there with any heuristic problem solving method.

As such the present approach has limited scope for strip layout problems because of its emphasis on optimum allocation position of a shape rather than its optimum orientation. Even in stock layout problems involving only single component this methodology may not suggest good layouts.

The restriction that the next allocation should always occur at the maximum y coordinate of the last allocated piece may sometimes leave waste space between two adjacent pieces.

CHAPTER VI

CONCLUSIONS

6.1 Summary

An approach has been described for optimal layout of both irregular as well as rectangular two-dimensional shapes into a rectangular stock using heuristic method. The algorithm produces an approximate solution which proves to be of good quality and efficient in terms of computer time. It works by hierarchically decomposing original problem into many sub-problems such that solution of each will imply a good solution of the original problem.

Four different modules for the solution i.e. creation of two-dimensional shapes, separation of tiny shapes, automatic nesting of all big shapes and interactive arrangement of tiny shapes, have been identified and implemented. All the modules are interactive and user-friendly.

The approach can be extended to solve problems where an irregular stock is considered and more complex constraints are included in the placement of the pieces. The program has been tested thoroughly for various examples. Its performance indicates that it favourably compares with that of Albano and Sapuppo 12 and is more efficient.

6.2 Scope for Future Work

Following modifications and additions in the approach are suggested for further work.

a) The algorithm can be modified to find all possible subsets (selected shapes sets) of a 'candidate shapes set' which can be feasibly allocated on the stock or a sub-part of it while satisfying all the constraints. The nearest neighbour approach can be applied to each subset in turn and that subset selected which produces the most optimum allocation. This will improve the results.

b) A tree-search approach can be used for finding optimum allocation for each shape in 'selected shapes set' which would allow backtracking and give better results.

c) The approach can be extended for solving three-dimensional allocation problems which are often met in applications like railway freight transportation and arrangement of crates in warehouses.

d) It can be extended for solution of problems for an irregular stock and subjected to more complex constraints.

e) The program can be altered to shift pieces along the y-axis so that they can be clubbed together along that axis too.

f) To reduce the 'Relative Waste' further, the last allocated right most shapes can be rotated by 90° and arranged along the width of the stock.

REFERENCES

1. Hinxman, "The trim-loss and assortment problems", Euro. Jour. of Opr. Res., Vol. 5, No. 1, July 1980, pp. 8-18.

2. Chow, "Nesting of a single shape on a strip", Int. Jour. of Prod. Res., Vol. 17, No. 4, 1979, pp. 301-307.

3. AYC Nee, "A micro-computer based blank layout solution for metal stamping", Sheet Met. Ind., March 1985, pp. 26-31.

4. Gilmore and Gomory, "A linear programming approach to the cutting stock problem", Opr. Res., Vol. 9, No. 6, Nov.-Dec. 1961, pp. 849-859.

5. Gilmore and Gomory, "Multistage cutting stock problem of two or more dimensions", Opr. Res., Vol. 13, No. 1, Jan.-Feb. 1965, pp. 94-120.

6. Christofides and Whitlock, "An algorithm for two-dimension cutting problems", Opr. Res., Vol. 25, No. 1, Jan.-Feb. 1977, pp. 30-44.

7. Adamowicz and Albano, "A solution of the rectangular cutting stock problem", IEEE Trans. on Sys. Man and Cyber;, Vol. 6, No. 4, April 1976, pp. 302-310.

8. Albano and Orsini, "A heuristic solution of the rectangular cutting stock problem", The Comp. Jour., Vol. 23, No. 4, Nov. 1980, pp. 338-343.

9. Beasley J.E., "Algorithms for unconstrained two-dimensional Guillotine cutting", J. Opr. Res. Soc., Vol. 36, No. 4, 1985, pp. 297-305.

10. Adamowicz and Albano, "Nesting two-dimensional shapes in rectangular modules", Computer Aided Design, Vol. 8, No. 1, Jan. 1976, pp. 27-33.

11. Albano, "A method to improve two-dimensional layout", Computer Aided Design, Vol. 9, No. 1, Jan. 1977, pp. 48-52.

12. Albano and Sapuppo, "Optimal allocation of two-dimensional irregular shapes using heuristic search methods", IEEE Trans. on Sys. Man and Cyber., Vol. 10, No. 5, May 1980, pp. 242-248.

13. Albano and Orsini, "A tree-search approach to the M-partitions and kanpsack problems", The Comp. Jour., Vol. 23, No. 4, Nov. 1980, pp. 256-261.

14. Dhande and Rumulu, "Computer aided methods for development of transition sections", Trans. of the ASME, Vol. 106, No. 3, Sep. 1984, pp. 401-408.

15. Rich E., "Artificial Intelligence", McGraw Hill Book Company.

16. Wilson and Farror, "Computation of geometrical and inertial properties for general areas and volumes of revolution", Computer Aided Design, Vol. 8, No. 4, Oct. 1976, pp. 257-262.

17. Aho, Hopcraft and Ullman, "Data Structures and Algorithms", Addison Wesley Publishing Company.

18. Nilsson, "Principles of Artificial Intelligence", Springer-Verlag, 1980,

ME-1987-M-MAH-COM